TURTLE-P : un profil UML pour la validation d'architectures distribuées

L. Apvrille [*] —— P. de Saqui-Sannes [,***] —— F. Khendek [*]**

[*] *Concordia University, Electrical and Computer Engineering Department, 1455 de Maisonneuve W., Montreal, QC, H3G 1M8, Canada*
{apvrille,khendek}@ece.concordia.ca
[**] *ENSICA, 1 place Emile Blouin, 31056 Toulouse Cedex 05, France*
desaqui@ensica.fr
[***] *LAAS-CNRS, 7 avenue du Colonel Roche, 31077 Toulouse Cedex 04, France*

RÉSUMÉ. Le profil TURTLE (Timed UML and RT-LOTOS Environment) étend les diagrammes de classes UML pour donner une sémantique formelle au parallélisme et aux synchronisations entre classes. Il ajoute aux diagrammes d'activité caractérisant le comportement de ces classes, des opérateurs de synchronisation et des opérateurs temporels (délai déterministe, délai non déterministe, offre limitée dans le temps). L'article propose d'étendre TURTLE dans l'optique d'une modélisation de protocoles et d'une validation d'architectures distribuées. Le nouveau profil est appelé TURTLE-P. Les diagrammes de composants et de déploiement sont inclus dans la méthodologie proposée. Une sémantique formelle leur est associée par traduction dans le langage RT-LOTOS. L'approche proposée est illustrée sur un système satellitaire. Le code RT-LOTOS dérivé des modèles TURTLE-P a été simulé et vérifié en utilisant l'outil RTL développé au LAAS-CNRS.

ABSTRACT. TURTLE (Timed UML and RT-LOTOS Environment) is a UML profile, which extends class diagrams to provide formal semantics for parallelism and synchronization between classes. Activity diagrams, which characterize the internal behaviour of classes, are enhanced with synchronization and temporal operators (deterministic delay, non deterministic delay, time-limited offer). This paper proposes to extend TURTLE for protocol modeling and distributed architecture validation purposes. Indeed, the new TURTLE-P profile includes a methodology which takes into account component and deployment diagrams. The formal semantics of these diagrams is given in terms of a mapping to RT-LOTOS. The proposed approach is illustrated through an application to a satellite system. The RT-LOTOS code derived from TURTLE-P models has been simulated and verified using the RTL tool developed at LAAS-CNRS.

MOTS-CLÉS : *UML, diagramme de déploiement, validation de protocole, RT-LOTOS.*

KEYWORDS: *UML, deployment diagram, protocol validation, RT-LOTOS.*

1 Introduction

Le nombre croissant d'applications distribuées soulève de nouveaux problèmes techniques (gestion des ressources distribuées, sécurité, etc.) mais aussi méthodologiques. Ainsi, si de nombreuses techniques de description formelles ont été proposées à des fins de modélisation de protocoles et de validation d'architectures distribuées, force est de constater que leur diffusion auprès des groupes de recherche et des industriels du domaine est loin d'être effective.

Parallèlement, de nombreux acteurs utilisent des approches beaucoup moins formelles mais qui deviennent des standards de fait. L'exemple type est la notation UML de l'OMG (*Object Management Group* [OMG03]). UML se pose en candidat naturel au support de conception de systèmes distribués [GOM00]. Pourtant, UML comporte de nombreuses lacunes, notamment en termes de sémantique formelle, d'opérateurs temporels et de méthodologie, ce qui compromet gravement la mise en œuvre d'une validation formelle d'un modèle UML. Pour répondre à cette problématique, nous avons proposé en [ASL01] un profil UML appelé TURTLE (*Timed UML and RT-LOTOS Environment*) qui s'est avéré efficace pour la conception de logiciels à contraintes temps-réel fortes [APV02] [LOH02]. Pour autant, l'adéquation de ce profil pour la modélisation de protocoles et de systèmes distribués peut être questionnée. En effet, si TURTLE permet effectivement de décrire le parallélisme et la synchronisation au sein d'une architecture de classes, puis de détailler le comportement temporel de ces classes, ce profil ne permet pas actuellement de modéliser de façon explicite l'architecture générale d'un système distribué et encore moins les paramètres de Qualité de Service. Pour remédier à ces insuffisances, cet article propose d'inclure dans TURTLE les diagrammes de composants et de déploiement. Le profil enrichi est appelé TURTLE-P.

L'article est structuré de la manière suivante. La section 2 donne une présentation synthétique du profil TURTLE. La section 3 positionne TURTLE par rapport aux travaux du domaine. La section 4 décrit l'extension proposée en termes de diagrammes de déploiement en y incluant la sémantique formelle obtenue par traduction vers RT-LOTOS. La section 5 traite de l'application de TURTLE-P à un système satellitaire. Enfin, la section 6 conclut l'article.

2 Le profil TURTLE

Le profil TURTLE (*Timed UML and RT-LOTOS Environment* [ASL01]) étend deux des diagrammes de la notation UML 1.5 [OMG03], à savoir les diagrammes de classes pour la description de la structuration (statique) du logiciel et les diagrammes d'activité pour la description du comportement interne aux classes. La sémantique formelle du profil est donnée par des algorithmes de traduction [LOH02] vers le langage formel RT-LOTOS [COU00], ce qui permet de réutiliser au bénéfice de TURTLE l'outil de validation RTL [RTL].

Un diagramme de classes TURTLE est constitué de classes « normales » et de classes stéréotypées appelées « *Tclass* ». Ces dernières peuvent se synchroniser et échanger des données au travers de portes de communication appelées *gates* (Figure 1a), qui se dérivent en *InGate* (porte d'entrée) et *OutGate* (porte de sortie). La synchronisation mais aussi le parallélisme entre *Tclasses* sont précisément définis par des opérateurs de composition représentés par des classes qui attribuent les relations d'association entre *Tclasses*. TURTLE emprunte à RT-LOTOS quatre opérateurs de composition: *Parallel* pour l'exécution en parallèle et sans communication (Figure 1b), *Synchro* pour la synchronisation par rendez-vous, *Sequence* pour l'exécution en séquence, et *Preemption* qui permet d'interrompre l'exécution d'une *Tclass* pour en activer une autre. TURTLE supporte également l'opérateur *Invocation* qui est le pendant de l'appel de méthode mais en mode de communication par rendez-vous. A ces cinq opérateurs de base sont venus s'ajouter deux opérateurs orientés « systèmes logiciels temps réel », à savoir *Suspend/Resume* et *Periodic* [APV02]. Notons que les opérateurs *Synchro* et *Invocation* doivent être accompagnés de formules OCL précisant les attributs de type *gates* mis en relation (par exemple {T1.g1=T2.g2}).

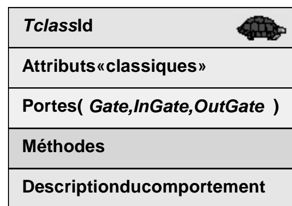

Figure 1a. Structure d'une *Tclass*

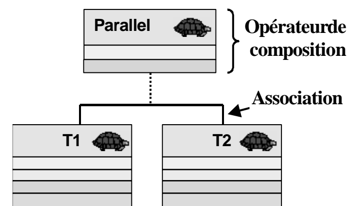

Figure 1b. Composition de deux *Tclasses*

Les diagrammes d'activité ont été étendus (Figure 2) pour exprimer les émissions et réceptions synchronisées, mais aussi des retards à durée fixe ou variable et des offres limitées dans le temps. Sur la Figure 2, la notation AD représente un sous diagramme d'activité interprété à la suite de l'opérateur.

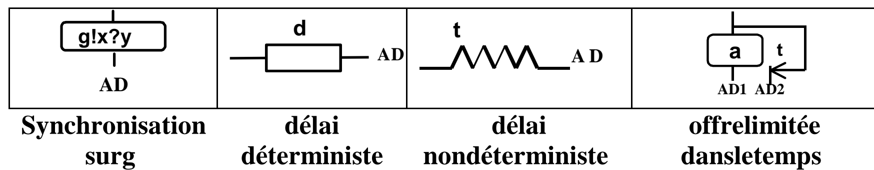

Figure 2. Nouveaux symboles pour les diagrammes d'activité

3 Positionnement par rapport aux travaux du domaine

La validation d'une architecture distribuée passe par la modélisation des protocoles de communication mis en œuvre dans cette architecture. L'idée d'expérimenter UML dans ce contexte est venue dès les prémices de la normalisation du langage UML à l'OMG. Aux exemples académiques traités en [JJP

98] et [JS 00] sont venus s'ajouter les protocoles de coopération [END 01], le commerce électronique [SG 01], les systèmes multi-agents [KKB 03] [LIN 01] et enfin ODP [HOL 97] [KMP 98] [BHK 00] [HUG 02]. La réutilisation de composants de communication devient un enjeu important [KMP98] [CAR00]. Ces travaux s'appuient dans leur ensemble sur la version normalisée d'UML. S'en tenir strictement à la notation de l'OMG est l'une des options défendues [DOU 02] dans le processus de «normalisation» d'un UML temps réel [WAT02].

Si les outils de validation et tests de protocoles développés antérieurement pour les techniques de description formelles s'avèrent réutilisables dans le contexte d'UML 1.4 [LEG 01], la notation de l'OMG a fait l'objet de plusieurs analyses critiques donnant lieu à des propositions de «profils» ciblés sur la validation d'architectures distribuées. Certains profils sont outillés. Ainsi, [WCW 01] décrit l'application à un protocole de commerce électronique du profil AUML utilisé conjointement avec le vérificateur de modèles *Spin*. AUML introduit des *Protocol Diagrams* qui étendent les diagrammes de séquences avec des opérateurs logiques, de causalité, synchronisation et diffusion. La validation avec *Spin* nécessite une traduction de ces diagrammes dans le langage *Promela*. Par ailleurs, [Sel 01] présente le profil implanté par l'outil Rose-RT de la société Rational. Rose-RT supporte un langage «cousin» de SDL [SDL], doté d'un seul opérateur temporel de type «délai fixe» et permet l'animation de modèles. Une autre approche appelée ACCORD/UML [GVK 00] s'avère plus riche en opérateurs temporels pour exprimer des échéances, périodes ou priorités au niveau des *Statecharts* UML.

Le profil TURTLE, introduit à la section précédente, dispose d'opérateurs temporels puissants (diagrammes d'activité) et donne une sémantique formelle au parallélisme et à la synchronisation entre classes (diagrammes de classes). Par contre, la définition donnée en [ASL 01] a laissé de côté les diagrammes de séquences et les diagrammes de déploiement. L'objectif du présent article est précisément de remédier à cette situation, en considérant plus particulièrement les diagrammes de déploiement. Notons que l'outil UML-AUT [LEG 01] supportait déjà les diagrammes de déploiement dans un environnement de conception de systèmes distribués mais nous n'avons pas trouvé en [LEG 01] trace de formalisation de ces diagrammes.

4 Le profil TURTLE-P

4.1 Description du déploiement de composants logiciels

Le diagramme de déploiement UML permet de mettre en évidence les configurations d'exécution d'un système. Ce diagramme est fort peu mis en valeur dans les méthodologies de développement de logiciels. De plus, il n'est utilisé qu'à des fins de documentation. Pourtant, nous pensons que ce diagramme est bien adapté à la description d'une architecture distribuée et nous voulons lui donner une sémantique formelle.

Un diagramme de déploiement est constitué de :

- ***Nœuds***. Ces nœuds constituent les différents sites physiques d'exécution du système considéré.
- ***Composants* logiciels**. D'après la norme UML, ces composants logiciels sont des éléments logiciels déployables qui encapsulent en leur sein des fonctions et qui offrent à leur environnement des interfaces.
- **Liens de communication**. Ces liens relient un composant logiciel à l'interface d'entrée d'un autre composant logiciel.

La Figure 3 présente un diagramme de déploiement UML dans lequel un composant logiciel *client* communique avec un autre composant logiciel *serveur* situé sur un autre site physique d'exécution.

Figure 3. Exemple d'un diagramme de déploiement UML.

Nous proposons d'améliorer le diagramme de déploiement UML comme suit :

- Les composants logiciels sont constitués de classes TURTLE et deviennent ainsi « validables » ;
- Une multiplicité peut être déclarée au niveau des nœuds afin de faciliter la modélisation des systèmes distribués comportant de nombreux nœuds identiques sur lesquels les mêmes composants sont déployés ;
- Les liens de communication peuvent être caractérisés par des paramètres (notamment, délai de transmission et gigue) qui sont pris en compte lors du processus de validation formelle.

4.2 Composants logiciels du diagramme de déploiement

4.2.1 Définition des composants

Un composant logiciel regroupe des classes qui réalisent une fonction logicielle sur un même site d'exécution. C'est pourquoi nous proposons de définir ces composants logiciels comme un sous-ensemble des classes modélisées dans le diagramme de classes. Soit C un composant logiciel tel que C soit un ensemble de n *Tclasses* = $\{c_i, i \in 1..n\}$. C doit respecter la propriété suivante :

> $\forall i \in 1..n, c_i \in C, \forall c \in$ *Diagramme de classes*, $c \notin C \Rightarrow$ *c et* c_i *ne sont pas en relation de Synchronisation ou d'Invocation.*

En effet, deux classes ne peuvent communiquer via une synchronisation ou une invocation que dans le cas où elles appartiennent toutes deux au même composant logiciel.

S’il existe entre deux *Tclasses* C1 et C2 d’un diagramme de classes, une relation de *Parallélisme*, *Séquence* ou *Préemption* et si un composant logiciel ne contient qu’une seule de ces deux *Tclasses*, alors la relation entre ces deux *Tclasses* est ignorée au niveau de ce composant. Notons aussi que deux composants logiciels distincts peuvent contenir la même *Tclass*.

Par exemple, considérons le diagramme de classes représenté à la Figure 4.

- C_a={C1,C3} ne constitue pas un composant logiciel car C1 et C2 sont en relation de synchronisation et C2 n’appartient pas à C_a.
- C_b={C1,C2,C4} est un composant logiciel valide car C4 s’exécute en parallèle au regard de C1 et C2 et aucune classe de C_b n’est en relation de Synchronisation/Invocation avec une classe n’appartenant pas à C_b.
- Cc={C3} est aussi un composant logiciel valide. C3 ne pourra jamais être préempté par C4 puisque C4 n’appartient pas à ce composant, mais cela ne «nuit» pas à l’exécution de C3.

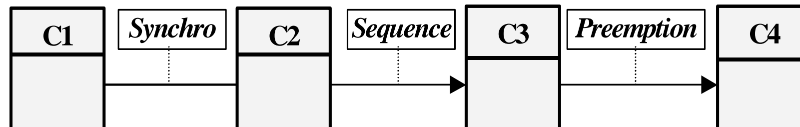

Figure 4. Exemple d'un diagramme de classes TURTLE.

4.2.2 *Interface des composants*

Les composants logiciels sont constitués de *Tclasses*. Pour communiquer, ces *Tclasses* offrent à leur environnement des portes de communication. Nous proposons que les interfaces des composants soient donc constituées des portes de communication des *Tclasses* de ce composant.

Cependant, certaines des portes de communication des *Tclasses* sont impliquées dans des relations de synchronisation à l’intérieur d’un même composant. Afin de ne pas pouvoir réutiliser ces mêmes portes avec des relations extérieures, seules les portes déclarées publiques (+) et non impliquées dans ces relations de synchronisation internes au composant constituent les interfaces des composants.

De plus, il nous faut distinguer les interfaces d’entrée des interfaces de sortie. Seuls les attributs de type *InGate* peuvent être considérés comme des interfaces d’entrée. De même, seules les portes de type *OutGate* peuvent être considérées comme des interfaces de sortie.

4.2.3 *Représentation des composants*

En UML, les composants logiciels représentés au niveau du diagramme de déploiement peuvent être définis au sein d’un diagramme de composants. Nous rendons cette démarche obligatoire.

Au niveau du diagramme des composants, seuls sont représentés les composants logiciels déployés au niveau du diagramme de déploiement et les interfaces des composants utilisées dans ce même diagramme de déploiement. Une porte *g* de la

classe *c* donnera une interface nommée *g_c*. Enfin, une multiplicité indiquée au niveau du nœud précise combien de fois l'ensemble des composants du nœud sont déployés sur des nœuds distincts. Par exemple, considérons le diagramme de déploiement représenté à la Figure 5. Le diagramme de composants correspondant est représenté à la Figure 6. Ce diagramme sert à mettre en évidence les classes logicielles contenues dans le composant, sans rappeler les liens sémantiques entre ces classes. La représentation des interfaces au niveau du diagramme des composants nous apparaît comme facultative dans la mesure où ces interfaces (c-à-d les portes des *Tclasses*) sont déjà spécifiées au niveau du diagramme de classes.

Notons que le nœud Client de la Figure 5 possède une multiplicité de *n* ce qui signifie que le composant C*b* est déployé sur *n* nœuds de type *Client*.

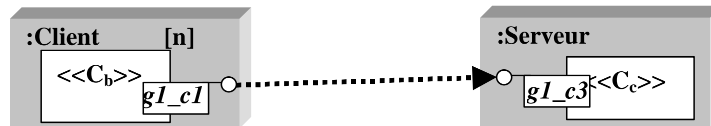

Figure 5. Exemple d'un diagramme de déploiement TURTLE-P

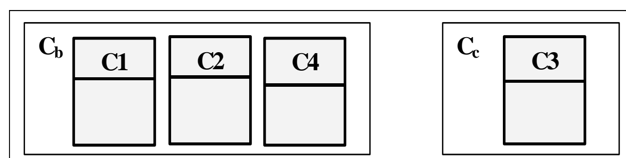

Figure 6. Diagramme de composants.

4.3 Liens du diagramme de déploiement

En UML 1.5, les liens des diagrammes de déploiement ne sont donnés qu'à titre d'information. Notre objectif est de leur donner une sémantique formelle.

Nous nous limitons ici à des liens de transmission asynchrone et unidirectionnels et attribués par quatre paramètres :

- un délai minimal (*delay_min*) et maximal (*delay_max*) de transmission ;
- une bande passante (*max_msg*) ;
- un taux de perte exprimé en pourcentage (*loss_rate*).

La Figure 7 représente un lien unidirectionnel entre un client et un serveur. La durée minimale de transmission sur ce lien est de 10 unités de temps, alors que la durée maximale de transmission est de 15 unités de temps. Au plus 100 messages peuvent être transmis simultanément sur ce lien. De plus, en moyenne 0,001 % des messages de ce lien sont perdus. La multiplicité de 10 précise que 10 clients peuvent cohabiter simultanément dans le système à l'exécution. Le fait d'introduire une multiplicité au niveau des nœuds soulève le problème du transit des messages : les messages entre un client et un serveur transitent-t-il sur le même lien que les messages entre un autre client et ce même serveur ? Par défaut, nous proposons que

les liens soient dupliqués i.e. que les messages ne transitent pas sur le même lien. Graphiquement, cette duplication de lien est identifiée par un disque évidé, cf. Figure 7. Dans le cas contraire où les messages transitent tous sur le même lien, ce lien unique est représenté graphiquement par un disque plein (cf. *Client2* sur la Figure 7).

De plus, un lien qui relie un nœud de multiplicité *n* vers un nœud de multiplicité *m* signifie que tout message émis sur ce lien est reçu par les *m* nœuds récepteurs. Si l'interface de réception du lien possède un disque plein, alors le message est considéré dupliqué à la réception. Sinon, il est dupliqué à l'émission.

Par exemple, dans le cas de la Figure 7, le nœud *Client1* est dupliqué à 10 exemplaires comportant chacun une instance du composant C_b, les instances communiquant avec le serveur via un lien indépendant. Par contre, les cinq composants C_b du nœud Client2 utilisent le même lien de communication pour envoyer un message au serveur.

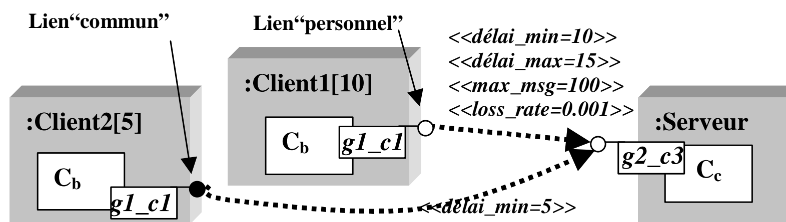

Figure 7. Modélisation d'un lien avec gigue et multi-clients.

4.4 Sémantique formelle des diagrammes de déploiement

Les algorithmes [LOH 02] de génération d'une spécification RT-LOTOS à partir d'une modélisation TURTLE acceptent en paramètre un diagramme de classes constitué de classes TURTLE dites *Tclasses* auxquels s'ajoutent des diagrammes de comportement inclus dans chacune de ces *Tclasses*. Nous proposons de nous appuyer sur les algorithmes définis dans TURTLE [LOH 02] dans le cadre de TURTLE-P.

La première étape de la traduction en RT-LOTOS consiste à transformer les diagrammes de déploiement en diagrammes de classes et d'activités TURTLE par les algorithmes de traduction de TURTLE-P : ce sont les nouveaux algorithmes que nous présentons ci-dessous.

Dans une deuxième étape, nous réutilisons les algorithmes de TURTLE pour générer la spécification RT-LOTOS à partir des diagrammes construits à l'étape précédente.

Les algorithmes de traduction TURTLE-P ne seront pas exposés in extenso compte tenu du nombre de pages alloué à l'article. Nous allons cependant donner une vue générale du processus de traduction.

Le diagramme de déploiement de TURTLE-P est un 2-uple $< N, L >$ avec N, une liste de nœuds $= \{n_i, i \in 1..n\}$ et L, une liste de liens $= \{l_i, i \in 1..m\}$.

Unnœudn $_i$estunt-uple< *C*, *mult*>avec *C*unensembledecomposants={c$_j$,j ∈ 1..n$_{ij}$}et *mult*lamultiplicitédunœud.

Unlienestun10-uple<n$_o$,n$_d$, *g1*, *g2*, *dmin*, *dmax,max_msg,loss_rate,type_o, type_d*>oùn$_o$etn$_d$dénotentrespectivementlesnœudsoriginesetdestinations, *g1*et *g2* dénotent les interfaces initiale et destination du lien, *dmin* et *dmax* représentent les délais minimum et maximum de transit des messages, *max_msg* le nombre maximaldemessagesentransitàuninstanttsurlelienet *loss_rate*letauxdeperte moyen du lien. Enfin, *type_o* ∈ { *lien_personnel*, *lien_commun*} et *type_d* ∈ {*dup_à_origine,dup_à_destination* }.

Nousnotons *Dc*lediagrammedeclassesTURTLEnatifquenousconstruisonsà partirdelamodélisationTURTLE-P.Cediagrammedeclassesestconstruitcomme suit:

```
//GénérationdesclassesdeDc
Pourchaquenœudn_i deN,pourchaquecomposantc_j deden_i
    Pourkvariantde1àlamultiplicitéden_i
        Les classes de c_j sont renommées de «nom_initial» à
«ni_k_cj_nom_initial»
        Lesclassesdec_j sontajoutéesàDc
    Fin
Fin

//Constructiondesliens
Pourtoutlienl_s=<n_op,n_dp,g_x_c_p,g_y_c_q,dmin_p dmax_p, max_msg_p,loss_rate_p,type_origine_p,
type_dest_p>deL
    Sitype_origine_p= lien_personnelalors
        Pourkvariantde1àlamultiplicitéden_op
            OnajouteàDcuneclasseLg_x_k_c_p telquelecomportementde
Lg_x_k_c_p reflètelecomportementdulieni.e.lagigue,labandepassante,letauxdeperteet
l'éventuelleduplicationdesmessagesavantouaprèstransmission(voiraprèsl'algorithme)
            On ajoute une relation de Synchronisation entre Lg_x_k_c_p et la
classec_q.LaformuleOCL{g_y}estajoutéeàl'association
        Fin
    Sinon
        On ajoute à Dc une classe Lg_x_c_p tel que le comportement de Lg_x_c_p
représentelelien
        Pourkvariantde1àlamultiplicitéden_op
            On ajoute une relation de Synchronisation entre la classe
ni_k_cj_cpetlaclasseLg_x_c_p.LaformuleOCL{g_x=g_k}estajoutéeàl'association
        Fin
        OnajouteunerelationdeSynchronisationentreLg_x_c_p etlaclassec_q.La
formuleOCL{g_y}estajoutéeàl'association
        Fin
    Fin
Fin
```

Finalement, un lien se caractérise par une relation de synchronisation entre, d'unepart,laclasseoriginedulienetuneclasseCdontlecomportementreprésente celuidulienet,d'autrepart,laclasseCetlaclassedésignéeparlelien.Iln'estpas possible de détailler dans le cas général le comportement de la classe C, mais simplement,parexemple,danslecasd'unlienentreuneclassec1,porteg1,etune

classe c3, porte g2. Le diagramme de déploiement correspondant est celui représenté à la Figure 5 (avec *n* qui vaut 1). Le diagramme de classes construit par les algorithmes précédents est représenté à la Figure 8. Le lien est ainsi modélisé par deux relations de synchronisation et une classe *Lg1_c1* dont le comportement émule celui du lien représenté à la Figure 5.

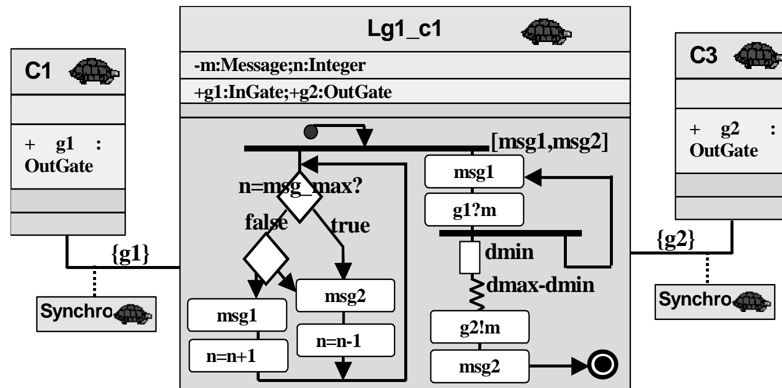

Figure 8. Représentation graphique en TURTLE d'un lien exprimé en TURTLE-P.

4.5 Méthodologie

Les diagrammes de composant et de déploiement sont intégrés dans une méthodologie décomposée en quatre étapes : (1) analyse du système, (2) conception, (3) conception détaillée et (4) validation (voir Figure 9).

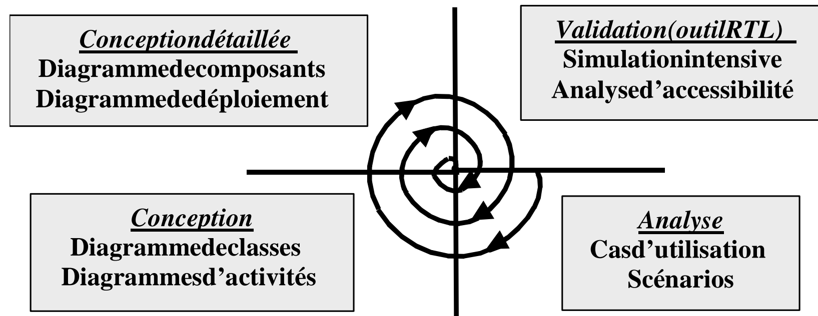

Figure 9. Méthodologie TURTLE-P d'analyse, conception et validation.

La phase d'analyse consiste à construire les cas d'utilisation du système et les scénarios d'échanges de messages, notamment entre les entités du système distribué.

La phase de conception consiste à construire le diagramme de classes du système et associer à chacune des *Tclasses* un diagramme de comportement.

La phase de conception détaillée utilise un diagramme de déploiement pour décrire la distribution des composants logiciels du système sur les différents sites

physiques d'exécution et les moyens de communication entre composants. De plus, par respect de la norme UML [OMG 03], les composants logiciels seront décrits dans un diagramme de composants qui fait partie intégrante de la méthodologie.

Cette méthodologie repose sur un cycle itératif et des raffinages successifs des diagrammes. Dans un premier temps, la conception « abstraite» et la conception détaillée se bornent à produire des diagrammes de classes et d'activités qui sont validés en utilisant l'approche initialement proposée en [ASL 2001]. Les diagrammes de composant et de déploiement sont réalisés dans un deuxième temps et la validation s'appuie une spécification RT-LOTOS obtenue en suivant le processus décrit à la section 4.4).

Dans une prochaine étude nous comptons élaborer une méthodologie de raffinement qui nous permettra de passer de la conception à la conception détaillée d'une manière semi-automatique. Cette méthodologie assurera par construction une relation de raffinement entre deux niveaux de spécification. Ce qui nous permettra de valider et de simuler le modèle au niveau conception pour certaines propriétés abstraites uniquement et de nous focaliser uniquement sur les propriétés spécifiques au déploiement lors de la validation de la conception détaillée.

5 Étude de cas

5.1 Aperçu du système

SAGAM [ROU 99] est un système de communication par satellite équipé d'un commutateur embarqué de cellules ATM qui permet de router les cellules entre les différents spots.

Une connexion utilisateur s'établit comme suit. Lorsqu'un utilisateur requiert une nouvelle connexion avec un autre utilisateur, le système évalue la charge requise par cette connexion (mécanisme de CAC) et en cas de succès, admet la connexion. Une fois cette connexion établie, l'utilisateur émet périodiquement ses cellules ATM dans des trames montantes. Afin que les slots alloués dans cette trame correspondent à son trafic réel (nous considérons du trafic VBR, donc variable), l'utilisateur peut émettre un signal, appelé *Dama-sig*, qui signale au système son désir de passer en mode ATM VBR-PCR (Peak Cell Rate). Le système repartit alors entre toutes les demandes la bande passante disponible et émet un plan de trame qui indique aux utilisateurs l'allocation des slots dans la future trame montante. Nous proposons dans cette étude de modéliser le mécanisme d'allocation de slots dans la trame montante.

Ce mécanisme est réalisé par trois entités distinctes : la station de l'utilisateur, le satellite et le centre de contrôle réseau du satellite (cf. Figure 10).

Au niveau de l'utilisateur (SU pour Station Utilisateur), une entité logicielle (Dama-client) permet d'émettre les Dama-sig et d'écouter en retour l'allocation attribuée. Le système satellite achemine alors les Dama-sig jusqu'à l'entité Dama Serveur située dans le centre de contrôle réseau du satellite (CR pour Centre

Réseau). Celui-ci évalue la demande et met éventuel lement à jour le plan de trame parémissiond'unsignalversl'entitéémettricede ceplandetrame,situéeàborddu satellite. Enfin, une entité logicielle à bord du s atellite reçoit les ordres de mise a jourduplandetramedepuisle«damaserveur»et émetunplandetrametoutesles 50millisecondes(entitéappeléePT).

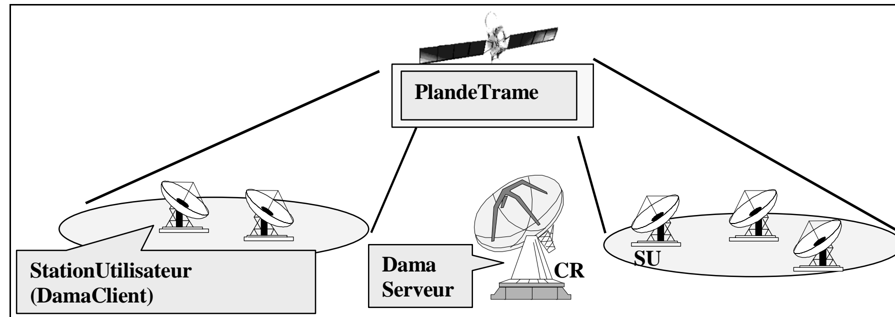

Figure10.Vuegénéraledusystèmeconsidéré.

5.2 ModélisationTURTLE-Pdusystème

Le système étudié a été modélisé en respectant les règles énoncées à la section 4.1.Celanousaconduità:

1. Décrireslediagrammedeclassesdusystème(non détailléici);
2. Extraire des composants logiciels de ce diagramm e de classes et les décrireauseind'undiagrammedecomposants.Cesc omposantssont:
   - o *DamaClient* constituédedeux *TclassesDc* et*Traffic* encharge des algorithmes du DAMA et du trafic généré au nive au de stationssols,respectivement.
   - o *DamaServeur* constitué de trois classes *Dsr*, *DamaAlgo*, *Dse* qui reçoivent, traitent et répondent aux Dama-sig, respectivement.
   - o *PT*constituédelaclasse *Ptm*quitraitelesdemandesdemises à jour du plan de trame reçues du *DamaServeur*, et *Pte* qui émetpériodiquementleplandetrame.
3. Réaliserlediagrammededéploiement(cf.Figure 11).

Outre les composants logiciels décrits précédemment , le diagramme de déploiementcomportetroisliensquiconnectentles troiscomposantssituéssurtrois nœudsdistincts.LenœudSU(StationUtilisateur)c omporteunemultiplicité, *n*,dont nousdiscutonsplusendétailparlasuite.

5.3 Simulationintensiveetvalidationformelle

L'outil RTL prend en entrée une spécification RT-LO TOS et permet de réaliser une simulation intensive ou de générer un graphe d' accessibilité. Dans le cas de

SAGAM, nous avons tenté une analyse d'accessibilité pour différentes valeurs de la multiplicité *n* (voir Figure 11). Pour des valeurs faibles de *n* (1 à 3), il nous a été possible de générer un graphe d'accessibilité comprenant un nombre d'états exploitable (de 195 états pour *n*=1 à presque 10000 pour *n*=3). Au-delà, l'approche par simulation intensive nous a paru plus adaptée.

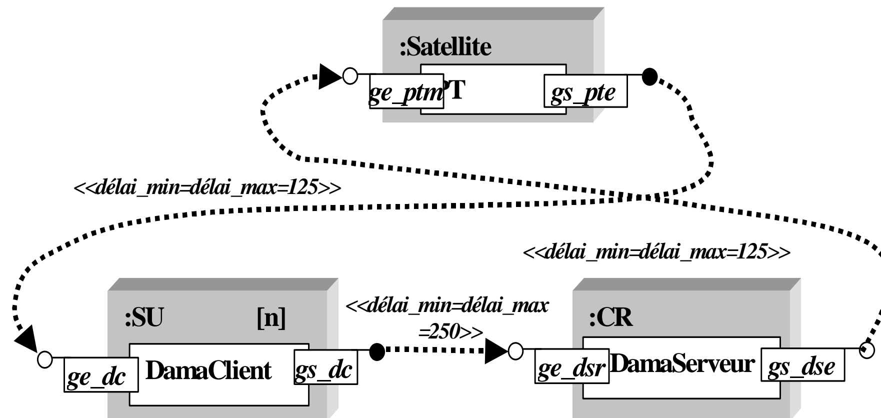

Figure 11. Diagramme de déploiement du système.

Outre l'absence d'états puits, la phase de validation a pour objectif de prouver le respect de propriétés au cours de l'exécution du système. Lors de contributions précédentes concernant le profil TURTLE, nous avons montré qu'il était possible, par insertion de classes TURTLE non-intrusives au niveau du diagramme de classes, d'analyser le respect de propriétés lors de la validation du système [APV 02][ASS 02]. Mais cette approche ne peut être utilisée qu'au niveau du diagramme de classes car elle consiste à ajouter un ou plusieurs observateurs en relation de Synchronisation avec les classes observées. Mais les liens de communication n'étant déclarés qu'au niveau du diagramme de déploiement, il n'est pas possible de leur adjoindre tels observateurs puisque la relation de synchronisation ne s'applique qu'entre des *Tclasses*.

Néanmoins, afin de résoudre le problème d'observation de propriétés sur les liens, nous proposons l'approche suivante. Considérons un lien tel que celui entre les composant *DamaServeur* et *PT* (cf. Figure 11). Dans le cas où l'on souhaite observer une propriété du lien qui nécessite l'observation d'un seul coté du lien (par exemple, savoir combien de messages ont été émis sur le lien), il suffit d'insérer un observateur dans le composant origine du lien, c'est à dire dans le composant *DamaServeur*, au niveau de la classe qui émet sur ce lien. Par exemple, dans le cas du lien entre *DamaServeur* et *PT*, la classe qui émet les messages est la classe *Dse* : il convient alors d'utiliser un observateur *O1* au sein du composant *DamaServeur* qui est en relation de synchro avec *Dse* et qui est capable de récupérer le nombre de messages émis sur la porte *gs* de la classe *Dse* (cf. Figure 12, partie gauche).

Dans le cas où la propriété observée nécessiterait de recueillir à la fois des informations à l'émission et à la réception du lien (par exemple, savoir combien de

messages sont actuellement sur le lien), il convient de placer un observateur à la fois au niveau de la classe émettrice, mais aussi au niveau de la classe réceptrice. Par exemple, toujours dans le cas du lien entre *DamaServeur* et *PT* (cf. Figure 11), il faut placer un observateur au niveau de la classe *Dse* et un au niveau de la classe *Ptm* (cf. Figure 12). Ainsi, l'observateur O1 note le nombre de messages entrants n_e et l'observateur O2, le nombre de messages sortants n_s. Pour calculer le nombre de messages sur le lien, il faut faire la soustraction entre ces deux nombres i.e. entre n_e et n_s. Pour cela, il convient que O1 envoie n_e à O2, ou que O2 envoie n_s à O1 : dans les deux cas, nous proposons l'insertion d'un lien de communication sans délai entre les deux observateurs au niveau du diagramme de déploiement (non représenté à la Figure 11).

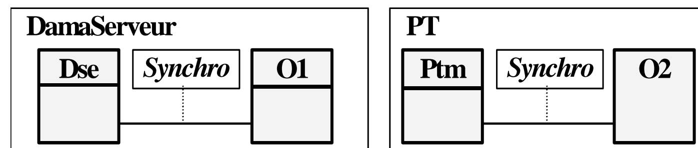

Figure 12. Observation de propriétés sur un lien : diagramme de composants.

Si la solution proposée pour observer une propriété est satisfaisante du point de vue résultat, elle n'en demeure pas moins fastidieuse à modéliser. Aussi, nous travaillons actuellement à la définition d'observateurs de liens (probes) qui permettront de simplifier la modélisation de l'observation de propriétés sur les liens. Le comportement de ces probes sera représenté au niveau du diagramme de classes tandis que le positionnement de ces probes sur les liens sera décrit au niveau du diagramme de déploiement.

6 Conclusion

Le profil TURTLE [ASL01] étend les diagrammes de classes UML pour donner une sémantique formelle au parallélisme et aux synchronisations entre classes. Il ajoute aux diagrammes d'activité caractérisant le comportement de ces classes, des opérateurs de synchronisation et des opérateurs temporels (délai déterministe, délai non déterministe, offre limitée dans le temps).

Si TURTLE a été appliqué avec succès à des systèmes temps réel [APV 02] [LOH 02], son adéquation à modéliser des protocoles et valider des architectures distribuées restait à expérimenter. Ce domaine d'application a motivé l'extension TURTLE-P décrite dans cet article qui ajoute à TURTLE les diagrammes de composants et les diagrammes de déploiement. Comme pour les diagrammes de classes et d'activités, la sémantique formelle de ces deux nouveaux diagrammes est donnée en termes de RT-LOTOS. L'exemple d'un système satellitaire illustre la méthodologie et en particulier la validation au moyen de l'outil RTL du LAAS.

Si TURTLE-P permet de modéliser un système réel du type de SAGAM traité dans l'article, il s'avère encore limité sur plusieurs points. Les premières extensions à apporter concernent la prise en compte sur les liens de nouvelles contraintes

(bande passante, ordre, taux de perte, etc.). Il serait également souhaitable de pouvoir utiliser un modèle de trafic au niveau des liens.

Dans la méthodologie proposée, les diagrammes de la phase d'analyse sont utilisés à des fins de documentation. A plus long terme, notre objectif est aussi d'utiliser ces diagrammes, soit pour synthétiser automatiquement le comportement des classes TURTLE comme cela peut déjà être réalisé pour les conceptions SDL [KGB 98][AKB 99], ou les utiliser comme scénarios de test lors de la phase de validation comme le permet déjà l'outil ObjectGeode [VER99].

7 Remerciements

Le profil TURTLE a été défini dans le cadre d'une coopération ENSICA-LAAS/CNRS impliquant Jean-Pierre Courtiat, Patrick Sénac et Christophe Lohr.

Nos remerciements s'adressent aussi à Michel Diaz (LAAS/CNRS) pour les idées qu'il nous a suggérées.

8 Bibliographie

[AKB 99] M. M. Abdalla, F. Khendek and G. Butler, "New Results on Deriving SDL Specification from MSCs", Proceedings of SDL Forum' 99, Montreal, Canada, June 1999.

[APV 02] L. Apvrille, «Reconfiguration dynamique de services logiciels de télécommunication par satellite», Thèse de doctorat de l'Institut National Polytechnique de Toulouse, juin 2002.

[ASL 01] L. Apvrille, P. de Saqui-Sannes, C. Lohr, P. Sénac, J.-P. Courtiat, "A New UML Profile for Real-time System: Formal Design and Validation", Fourth Intern. Conference on the Unified Modeling Language (UML'2001), Toronto, Canada, October 2001.

[ASS 02] L. APVRILLE, P. DE SAQUI-SANNES, P. SENAC, C. LOHR, «Reconfiguration dynamique de protocoles embarqués à bord de satellites», *Actes du Colloque Francophone sur l'Ingéniérie des Protocoles* (*CFIP'2002*), Montréal, Canada, Mai 2002 (Hermès ed.).

[BHK 00] M. Born, E. Holz, O. Kath, "A Method for the Design and Development of Distributed Applications Using UML", TOOLS-Pacific 2000, Sydney, Australia.

[CAR 00] E. Cariou, «Spécification de composants de communication en UML», OCM 2000, Nantes, France, mai 2000.

[COU 00] J.-P. Courtiat, C.A.S. Santos, C. Lohr, B., Outtaj "Experience with RT-LOTOS, a Temporal Extension of the LOTOS Formal Description Technique", *Computer Communications*, vol. 23, n. 12, 2000, p. 1104-1123.

[DOU 02] B.P. Douglass, "Real-Time UML Tutorial", OMG Real-Time and Embedded Distributed Object Computing Workshop, Arlington, VA, USA, July 2002.

[END 01] J.M.M. Espinosa, O. Nabuco, K. Drira, "A UML Model for Session Management in Collaborative Design for Space Activities", 8th European Concurrent Engineering Conference (ECEC'2001), Valence (Espagne), pp. 170-174, April 2001.

[GOM 00] H. Gomaa, "Designing Concurrent, Distributed and Real-Time Systems with UML", Addison Wesley, ISBN 0201657937, 2000

[GVK 00] S. Gérard, N.K. Voros, C. Koulamas, F. Terrier, "Efficient System Modeling of Complex Real-Time Industrial Networks Using the ACCORD/UML methodology",

DIPES 2000, International IFIP WG10.3/WG10.4/WG10.5 Workshop on Distributed and Parallel Embedded Systems, Paderborn, Germany, October 2000.

[HOL 97] E. Holz, "Application of UML within the Scope of new Telecommunication Architectures", Humboldt-Universität zu Berlin, 1997.

[HUG 02] M.-P. Huget, "Extending Agent UML Protocol Diagrams", Agent Oriented Software Engineering (AOSE-02), Fausto Giunchiglia and James Odell and Gerhard Weiss (eds.), Bologna, Italy, July 2002.

[JJP 98] C. Jard, J.-M. Jézéquel, F. Pennaneach, «Vers l'utilisation d'outils de validation de protocoles dans UML», Techniques et Sciences Informatiques, v.15, n°11, pp.1-15, 1998.

[JS 00] M. Jaragh, K.A. Saleh, "Modeling Communications protocols using the Unified Modeling Language", TENCON'2000, Intelligent Systems and Technologies for the New Millenium, Kuala Lumpur, Malaysia, September 2000.

[KGB 98] F. Khendek, R. Gabriel, G. Butler and P. Grogono, "Implementability of Message Sequence Charts", Proceedings of the first SDL Forum Society Workshop on SDL and MSC, Berlin, Germany, June 29-July 1, 1998.

[KKB 03] K. Kavi, D.C. Kung, H. Bhaambhani, G. Pancholi, M. Kanikarla, R. Sah, "Extending UML to Modeling and Design of Multi-Agent Systems"; submitted to the International Conference on Software Engineering, 2003.

[KMP 98] M. M. Kandé, S. Mazaher, O. Prnjat, L. Sacks, M. Vittig, "Applying UML to Desin an Inter-Domain Service Management Application", UML'98, Mulhouse, France, June 1998.

[LEG 01] A. Le Guennec, «Génie logiciel et méthodes formelles avec UML : spécification, validation et génération de tests», doctorat de l'Université de Rennes I, juin 2001.

[LIN 01] J. Lind, "Specifying Agent Interaction Protocols with Standard UML", Second International Workshop on Agent-Oriented Software Engineering (AOSE-2001) Volume 2222 of Lecture Notes in Computer Science, Springer Verlag, Heidelberg, March, 2002

[LOH 02] C. Lohr, «Contribution à la spécification des systèmes temps réel en s'appuyant sur la technique de description formelle RT-LOTOS», thèse de Doctorat de l'Institut National Polytechnique de Toulouse, décembre 2002.

[OMG 03] OMG, "Unified Modeling Language Specification", Version 1.5, Object Management Group, http://www.omg.org/technology/documents/formal/uml.htm

[ROU 99] L. Roullet, "SAGAM Demonstrator of a G.E.O. Satellite Multimedia Access System: Architecture & Integrated Resource Manager", European Conference on Satellite Communication, Toulouse, France, November 1999.

[RTL] http://www.laas.fr/RT-LOTOS

[SEL 01] B. Selic, «A UML Profile for Modeling Complex Real-Time Architectures», http://www.omg.org/news/meetings/workshops/presentations/realtime2001/6-3%20Selic.presentation.pdf.

[SDL] http://www.sdl-forum.org/.

[SG 01] I.W. Siu, Z.S. Guo, "The Secure Communication Protocol for Electronic Ticket Management System", University of Macau, June 2001.

[VER 99], Verilog, "ObjectGeode", Toulouse, France, 1999.

[WAT 02] B. Watson, "The Real-Time UML Standard", OMG Real-Time and Embedded Distributed Object Computing Workshop, Arlington, VA, USA, July 2002.

[WCW 01] J. Wei, S.C. Cheung, X. Wang, "Exploiting Automatic Analysis of E-Commerce Protocols", 25th Annual Computer Software and Application Conference, Chicago, USA, October 2001.